TITLE OF THE INVENTION

NETWORK FILESYSTEM ASYNCHRONOUS I/O SCHEDULING

CROSS-REFERENCE TO RELATED APPLICATION(S)

[0001] This application is related to EQUITABLE RESOURCE SHARING BETWEEN LOCAL AND NETWORK FILESYSTEMS, filed concurrently by David Chinner and Michael A Gigante and incorporated by reference herein.

BACKGROUND OF THE INVENTION

1. Field of the Invention

[0002] The present invention is directed to processing of filesystem resource acquisition requests or access requests and, more particularly, to avoiding monopolization of access to a filesystem by one type of filesystem access request.

2. Description of the Related Art

[0003] A network filesystem, such as NFS from Silicon Graphics, Inc. (SGI) of Mountain View, California, provides access to files in centralized storage by client computer systems in a client-server operating environment. Such network filesystems process resource acquisition requests for read and write access to data and to obtain metadata or information about the data, such as the size of the data, access permissions on the data, etc.

[0004] Conventionally, resource acquisition requests are stored in one or more queues upon receipt while they await execution by a process that controls access to the filesystem. When more than one queue is used, the resource acquisition requests may be sorted by read, write and metadata requests and, and by different subtypes within these types, e.g., within read requests by whether a read-ahead operation should be performed, or whether a write contains data or only a synchronization request.

[0005] Regardless of whether all resource acquisition requests are stored in a single queue, or sorted into multiple queues, a single type of resource acquisition request may monopolize access to a filesystem, or a particular type of resource acquisition request which is desired to be given higher priority may be slowed by execution of less important resource acquisition requests. For example, an application may generate a sequence of write operations that monopolize a filesystem if requests in one queue at a time are executed until the queue is exhausted or there is only one queue, or a read of a block of data may be delayed by less important

metadata and write access requests in a system which services the queues in a round-robin fashion. No known system is flexible enough to allow users, i.e., system administrators, to tune the processing of resource acquisition requests for a particular operating environment.

SUMMARY OF THE INVENTION

[0006] It is an aspect of the present invention to prevent one type of filesystem resource acquisition request from monopolizing access to a filesystem.

[0007] It is another aspect of the present invention to prevent less important filesystem resource acquisition requests from slowing execution of more important filesystem resource acquisition requests.

[0008] It is a further aspect of the present invention to prevent filesystem resource acquisition requests that have no latency requirements from slowing the execution of filesystem resource acquisition requests that require minimal latency.

[0009] It is yet another aspect of the present invention to provide configurable metering for different types of filesystem resource acquisition requests.

[0010] The above aspects can be attained by a method of processing resource acquisition requests, including scheduling execution of the resource acquisition requests in accordance with user configurable metering. Preferably, the resource acquisition requests are sorted into queues with at least one queue for read requests and at least one queue for write requests. At least one other queue may be provided for metadata resource acquisition requests. Preferably, metering of the resource acquisition requests is configured in response to input from an administrator of the system., by specifying a first number of read requests to be performed for a second number of write requests, as long as there are both read and write requests that are queued. Preferably, a maximum number of threads are established for executing resource acquisition requests in response to the input from the administrator.

[0011] These together with other aspects and advantages which will be subsequently apparent, reside in the details of construction and operation as more fully hereinafter described and claimed, reference being had to the accompanying drawings forming a part hereof, wherein like numerals refer to like parts throughout.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a block diagram of a system to which the present invention can be applied.

Figure 2 is a flowchart of a method according to the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0012] A simplified drawing of a system to which the present invention may be applied is illustrated in Fig. 1. Client node 10 is connected via network 12 to at least one filesystem server 14a, 14b, ... 14n on network 12. Depending on the type of system, each filesystem server 14 may provide connection to one or more data stores 16. In some systems, such as storage area networks, the network 12 may be directly connected to the data stores 16a, 16b, ... 16n. Client node 10 uses one of client network filesystems 18a, 18b, ... 18n in accessing each data store 16. Client node 10 may be connected to many data stores, e.g., 16a, 16b, ... 16c, on a single filesystem server, e.g., 14a, as well as many separate data stores, e.g., 16a, 16d, 16f on many different filesystem servers, e.g. 14a, 14b, ... 14n.

[0013] Any communication requests generated by a client network filesystem, e.g., 18a, will be directed to the filesystem server 14 with responsibility for the data store(s) 16 storing the data for filesystem 18a. These requests are often asynchronous and client network filesystems 18 can generate large numbers of requests in a short period of time. To optimize processing of these communication requests, each client network filesystem 18 has a set of request queues 20a, 20b, ... 20n that store requests that have not been sent to the corresponding filesystem server, e.g., 18a. These queues are serviced by a scheduler and set of execution threads 22a, 22b, ... 22n which are responsible for determining the best request to send and executing the communication procedures to fulfill the request.

[0014] As illustrated in Fig. 2, client node 10 receives configuration data 22 which is used to set a ratio N:M of N read access requests that should be processed for M write access requests. In addition, the number of threads used to process the access requests may be set 24.

[0015] Metadata requests are considered to be both of higher priority and more latency sensitive than either read or write requests. They are also unlikely to occur in such numbers as to cause read or write starvations. As a result, preferably the metadata request queue should given an infinite quota and hence the ratio of metadata:read:write requests when all are queued is infinity:M:N. In practice, enough metadata requests are not received for this to cause starvation of the read or write queues. Hence if, at any point in time, a metadata request is queued waiting to be executed, then the next available execution thread will process the metadata request ahead of any queued read or write requests.

[0016] When resource acquisition requests 26 are received by file system server 14, the resource acquisition requests are sorted 28 into queues. In a system executing under IRIX, where little improvement is found from sorting read requests into different categories or write requests into different categories, three queues are preferably used, one for read requests, one for write requests and one for metadata requests. Under other operating systems, additional queues (or just two queues) could be used. For example, an open network computing (ONC) NFS available from Sun Microsystems, Inc. of Santa Clara, California has five queues, one for read-ahead requests, another for read requests, one for write requests, another for write synchronization requests and one for directory read requests. In the ONC NFS, requests are taken from the queues in a round-robin fashion. A number of consecutive requests are taken from each queue before moving on to the next queue.

[0017] When an access request occurs, it is first sorted into the relevant queue and a execution thread is signaled to run. The first idle execution thread will then run the request quota scheduling 30 to the sorted queues in accordance with the ratios previously set 24 by the administrator of the system to select the request it is to execute.

[0018] The execution thread then initiates a series of communications to fileserver 18 to transfer the required data to fulfill the current request. This can require multiple communications and take a significant period of time. Once the request has been completed, the execution thread will mark the request as completed, and return to the sorted queues to determine whether there is another request to execute. If there are no more requests to execute, the execution thread will then sleep until new requests to process have arrived in the sorted queues.

[0019] The present invention has been described with respect to an embodiment using SGI equipment running NFS under IRIX. However, it is not limited to such an operating environment and can be used with many different filesystems. For example, it could be used in the aforementioned ONC NFS implementation to improve it. The invention could also be used on an NFS server to prioritize the order of responses to different types of requests to improve the response time of a fileserver to latency sensitive requests without causing starvations in different layers of the server. In the NFS layer the invention could be used to expedite metadata requests over reads and writes to a single client. In the network interface layer the invention could be used to expedite small responses (typically metadata responses) over large responses (typically read responses) between all traffic that is routed through the network interface.

[0020] For a general case of n queues, every queue (q1, q2, ..., qn) has a quota (r1, r2, ... rn) that determines the number of requests that can be issued from that queue while there are other requests held on other queues and the quota for those queues is not exhausted, and every queue has a importance rating. The importance rating determines the order in which queues are scanned for held requests and remaining quotas. The next request to be executed is taken from:

(A) the highest importance queue that has not exhausted it's quota and has a held request.

(B) the highest importance queue that has exhausted it's quota and has a held request if and only if there are no lower importance queues that have also exhausted their quota and have held requests.

[0021] Condition (A) ensures that when all request queues have pending requests, the ratio of requests as configured by the administrator is always maintained. Condition (B) allows requests to be executed even if condition (A) is not met to prevent unnecessary delays when other types of requests are not being issued. This allows request queues to operate at full speed when other queues are not being used.

[0022] Given this, a queue with a higher importance rating will tend to exhaust its quota sooner than a queue with a lower importance, but condition (B) will allow it "free" executions if there are no lower importance requests being held. It is desirable to give latency sensitive request types higher importance ratings due to this behavior.

[0023] Quotas are refreshed when:

(C) the quota for every queue reaches zero; or

(D) more than one queue has exhausted its quota and two or more of these queues have held requests.

[0024] Condition (C) indicates that all quotas are exhausted and hence the quotas for each queue can be reset and the cycle can begin again. Condition (D) indicates that there is more than one type but not every type of request being issued. This is a situation that condition (B) would lead to one queue dominating execution by being continually granted "free" executions but processing needs to continue ensuring that the configured ratio of requests is observed. Hence the quotas are reset to enable this. In essence, this provides a method for enforcing a

quota only when it needs to be enforced. This improves both throughput and latency and prevents starvations when one or more types of requests dominate the others.

[0025] In the example described above, only read and write quotas are configured because the metadata queue is given an infinite quota. In other words, metadata requests are considered to be both more important and more latency sensitive than either reads or writes. As a result the algorithm described above is simplified significantly, so that the system administrator can be requested to configure a simple M:N read:write ratio. In actuality there is a L:M:N metadata:read:write ratio where L is fixed at infinity.

[0026] With reference to the conditions above, in the described example if there is a metadata request held on the queue, then condition (A) will always execute it first. If there are no metadata requests held, then reads or writes are processed depending on condition (A) or (B). It also means that the quota reset occurs when condition (D) comes true, as condition (C) never occurs due to the metadata quota being infinite.

[0027] The many features and advantages of the invention are apparent from the detailed specification and, thus, it is intended by the appended claims to cover all such features and advantages of the invention that fall within the true spirit and scope of the invention. Further, since numerous modifications and changes will readily occur to those skilled in the art, it is not desired to limit the invention to the exact construction and operation illustrated and described, and accordingly all suitable modifications and equivalents may be resorted to, falling within the scope of the invention.